Visualization and Pre-Processing of Intensive Care Unit Data Using Python Data Science Tools

Jacob Barhak

Austin, TX
jacob.barhak@gmail.com

ABSTRACT

Abundance of information is collected in the Intensive Care Unit (ICU) during patient stay. To help humans cope with abundance of such data, computers can help with 1) pre-processing and 2) visualization. Fortunately modern languages such as Python provide good tools to help programmers aid in those tasks. Tools such as Pandas allow import of data into a standard dataframe that can be processed with machine learning algorithms using scikit-learn, and later this data can be visualized interactively in web environment using Bokeh. The Jupyter notebook interface allows quick prototyping of the methods. This work shows how this collection of tools can be used to visualize patient data and filter it for outliers using multiple machine learning algorithms. The prototype created in this work is applied on a public ICU dataset with the intention of future extension towards automated detection of outliers in the data. Such outliers can either indicate hazards or should be removed before further analysis. Open source code and visualization are publicly available to support this paper.

ABOUT THE AUTHORS

Jacob Barhak specializes in chronic disease modeling with emphasis on using computational technological solutions. Dr. Barhak has diverse international background in engineering and computing science. He received his Ph.D. in 2003 from the Technion Israel Institute of Technology. From 2003 to 2006 he worked at an NSF Engineering Research Center at the University of Michigan focusing on inspection systems and personalization. From 2006 to 2012 he worked on developing disease modeling tools for the Michigan model for diabetes at the University of Michigan. The Reference Model for disease progression was independently self-developed by Dr. Barhak in 2012. He is the developer of the MIcro Simulation Tool (MIST).

Visualization and Pre-Processing of Intensive Care Unit Data Using Python Data Science Tools

Jacob Barhak

Austin, TX
jacob.barhak@gmail.com

INTRODUCTION

The amount of information collected today about a patient is growing rapidly supported by Electronic Medical Record (EMR) systems (Olchanski et al 2013). Moreover, medical databases such as ClinicalTrials.Gov (Zarin, et al ,2016), (Ide et al , 2007) are growing in size and provide valuable data that is constantly growing. This growing data raises many issues such as data sharing / ownership (Gore & Chandra 2015), and puts pressure on modeling to provide proper analytics.

Fortunately, scientific data tools are constantly improving and are made accessible to the public. More specifically, the Python Language and its scientific tool set is being made freely available. Tools such as Pandas for handling data, Jupyter notebook for rapid prototyping, Scikit-learn (Pedregosa et al, 2011) , (scikit-learn, Online) for machine learning and Bokeh (Bokeh, online) for visualization are free and available and bundled together in the Anaconda scientific Python distribution. Such tools are used for many data analytics tasks and can handle large amounts of data, especially with the availability of computing power in cloud platforms, supporting High Performance Computing (Star Cluster, Online).

However, despite the availability of modeling tools and the growing amount of medical data, not enough work has been done to apply analytics on medical data, despite the great potential. Part of the issue is because the data was previously scattered and many times not in electronic format. However, even data in electronic format is not necessarily accurate and even accurate data needs some pre-processing. Specifically of interest in this paper is Intensive Care Unit (ICU) data.

ICU data processing is increasingly becoming popular for processing. This can be seen in papers such as (Celi & Mark 2013) that describe multiple sources of data for researchers to access. Perhaps the more known data source comes from the physionet project that holds several databases (PhysioNet, The MIMIC Database, Online) , (PhysioNet, MIMIC II, Online) , (PhysioNet, The MIMIC-III Clinical Database, Online) that can be accessed online.

This work started with the idea of using analytics to automate prediction/projection using the National Early Warning Score (NEWS) (Royal College of Physicians ,Online). Having an improved mechanism to automatically detect patients that are at risk is highly desirable. However, before improving projective analytics, the first step of data processing is normalizing the data and visualizing it. Only this preliminary processing alone becomes a challenge, even when processing data from a single patient.

Data of one such patient was posted publicly on (UCI, Machine Learning Repository, Online) (Lichman, 2013) for a conference. This data is the focus in this work that attempts to perform the very basic tasks of:
1) Data Import – including cleaning and normalization.
2) Data Visualization – with the intent of explaining phenomenon that will improve understanding (Tufte, 1997).
3) Detecting outliers – to provide visual cues to the human viewer.

This paper will show that even those seemingly simple tasks are non trivial and require some effort even before more sophisticated data analytics is applied on multiple patient data sources. It is important to note that the paper is written from the perspective of a computational modeler and not from a clinical perspective. The author invites feedback from clinicians and other data analytics to improve this work. To help this, the code and data associated with this paper are publicly available in Jupyer notebook format in https://github.com/Jacob-Barhak/VisICU

DATA IMPORT

The data used in this paper was provided as 7 text files in various formats. The files are described in Table 1. Unfortunately some of the files were clearly manipulated by human hand – probably during data entry, so some minor fixes were needed to clean up the data for some files. Table 1 describes how each file was manipulated. Fortunately Python has good tools to allow importing such data using Pandas and regular expressions. The Pandas dataframe is a common database to help convenient data manipulation. This data set is old which may explain inconsistencies in format. However, even with newer, more organized data it is not uncommon to have to process and clean up data for processing.

The data set contained 3 main measurements sources: 1) Flowsheet, 2) Lab data 3) Monitor data. All those data sets were normalized to be included in the same dataframe that includes: Date/time, Code, Name, Unit, and Origin for each measurement in all files using the same format for the date/time – to simplify querying and visualization.

Table 1. Data set description and clean up

Original File	Format	Comments	Clean Up
Domain-Description	Free Text	The file contains free text describing the problem and its background. It also contains Lab data code description at the end that is important to understand other files.	The lab data was copied and edited by hand to become a tab separated file containing a table with the name: Domain-Description-Edited.txt this file can be imported as tab separated data by Pandas.
Flowsheet-Data	Tab Separated / Space Separated	The file contains flowsheet values along a timeline including description and units. The data is provided as tab or space separated values mixed.	Regular expressions were used to parse the text file to determine column separation that will keep date components together and ignore differences in data entry between lines and will generate 3 columns in output. time, description, and value.
Lab-Data	Fixed Width	The data was well organized as fixed width columns containing code, name, and units. The code is expanded to text in the Domain Description file.	Imported using Pandas fixed width format and space trimming around the code field.
Monitor-Data	Tab Separated	Described monitor data with Time, code and data. However, date is not included in the time and recording starts around 10pm lapping midnight.	Imported using Pandas separated values import with special attention to the data/time format. The date component was added to the time such that day times before 10pm, when recording started, were considered the day after start.
Monitor-Data-Codes	Tab Separated	Describes the monitor data code meanings by providing description and units. The first definition is split across several lines which makes import not trivial for each line.	The file was edited manually to merge the first few lines together to have one code per line. The modified file Monitor-Data-Codes-Edited.txt was then imported using Pandas separated values import
Patient-Description	Free Text	A file providing brief description of the patient.	Not imported
README-ICU	Free Text	Wrapping file describing the data set alongside origin.	Not imported

VISUALIZATION

Rapid prototyping Python techniques were used during development within Jupyter Notebook environment that also allows incorporating visual data that can be viewed through a web browser. The graphical framework chosen for visualization was Bokeh. Bokeh supports the creation of html files that can be viewed online. It also supports creating a local Bokeh server that will support viewing graphics through a web browser on an offline computer. In both online and offline cases, there is a Bokeh server that allows interactive visualization such as mouse hovering popups, legend clicking, and even animation or widgets such as buttons and sliders. In this work the data is displayed in one html file located online at http://sites.google.com/site/jacobbarhak/home/VisualICU.html . The file contains multiple plots with limited interaction consisting of mouse hovering and legend show/hide by clicking.

The main challenge in visualizing the file was abundance of information. The data contains 7931 monitor records from 13 different categories of measurements, 81 lab data measurements from 38 categories and 38 flowsheet data records. This abundance of data collected over less than a day is hard to categorize or view. The idea is to allow the viewer to see as much information as possible in one location. However, there are multiple types of measurements originating from different sources which have different scales and units and different meanings. The compromise that was found reasonable is governed by the following concepts:

1. All plots of data will be presented using the same time scale – typically plots are presented in one column where the horizontal axis in all plots is similar.
2. If a specific measurement has more than 10 data points, it is considered as a major time series plot and therefore be shown on its own, otherwise it will be aggregated with other measurements from the same source. The aggregated sources are titillated as: 1) Other Flowsheet, 2) Other Monitor, and 3) Other Lab. These are provided at the end of the plot set.
3. Each data point has additional information such as description and/or unit. There is not enough space to present this information in each plot, especially in the aggregated plots, so this information is displayed only if the user requests it by hovering the mouse over the data point.
4. Each measurement in the time series is represented with different color. Since there are not enough colors to visually distinguish between measurements, and we still want points from the same measurement to look the same, two colors are used to represent a data sample, fill color and outline color. This provides enough combinations to visually distinguish between most measurements.
5. The major time series plots contain outlier information. So the filtering plot is provided to the right of the data plot for information on how filtering is performed. On the main plot the color of each point corresponds to the level of filtering. The filtering plot to the right shows multiple methods of filtering and aimed mostly for development purposes where the developer can show/hide specific series by clicking of the corresponding legend. On the filtering plot to the right, each series corresponding to a filtering method uses a different color to help with this exploration, while the left plot shows the summary information.

With this visualization it is possible to view the entire information on one scrolling window which quickly visually reveals a few elements even to the untrained eye:

1. First lab data reported much before flowsheet data and monitor data arrives last – this may reveal information about the procedure the patient went through.
2. Lab data was taken 6 times while the flowsheet was visited a bit more frequently since some measurements have 15 minutes difference.
3. Some data outliers are clearly visible to the untrained human eye. The question is whether those outliers should be filtered away from the data as bad measurements or whether they indicate important information that the physician should be alerted to. In either case, we wish the machine to mark those points, either for future removal or future alert.

The difficulty of visualizing this abundance of information for one patient for a relatively small period of time shows the importance of machine analysis, specifically the outlier issue seems a place where machine learning algorithms can help outlier detection.

OUTLIER DETECTION

There are 10 time series in the data with 10 or more data points which was considered sufficient data to be considered major time series. However, there is much variability among those, the number of sample point varies from 13 to 1984 points. Also, the measurements have different scales and characteristics, some measurements are clearly discrete even boolean in values, while others seem to have more value bins, and some seem to be on a continuous analog scale. Moreover, some measurements seem to arrive on most time intervals while others have larger and irregular gaps on the time axis. Even an untrained human looking at those series can visually detect what seems as outliers. The goal in this work was to try and let the machine figure out those outliers in mostly unsupervised manner.

Since no prior information on the measurement was assumed, nor examples from other patients for the same measurements were available, and since no expert human classifications were available, machine learning techniques that are based on training were excluded from computation. However, to handle the variety of measurement data forms, it was decided to use more than one technique to detect outliers. Seven classification and filtering techniques were used, mostly from the Python scikit-learn library. Then each measurement point was voted on by different techniques to give it an outlier vote level. The color of each point in the visualization left column of plots is determined by the number of votes for each point of being an outlier. Hovering with the mouse over the point will show the number of votes that point received from the seven different filtering methods.

To streamline the process, each filtering technique received only the points associated with a specific measurement from the Pandas dataframe and the data was normalized so that the time scale and the point scale will fit the [0,1] bounds. Note that this may squeeze stretch the data in different direction and therefore change gap sizes, which may affect filter algorithm results. However, this is an important step towards standardization of the data and is essential for many methods to work properly. Once data was normalized, the filtering techniques below were applied.

Filtering Using Gaussian Process

This is a statistical regression technique that is implemented in scikit-learn using the GaussianProcessRegressor method. The technique documentation is available in http://scikit-learn.org/stable/modules/gaussian_process.html . Another easy introduction to the technique can be found in (Fonnesbeck, 2017). In a nutshell, the technique assumes normal distribution and tries to measure the similarity between points assuming normal distribution, and from these project the next point. The similarity measure is determined by providing a kernel function that can be constructed from multiple components. In this implementation the kernel consisted of a constant value, Matern kernel, and a white noise component. Another value in input, designated as “alpha”, controls the noise level estimate. This technique is sensitive to data normalization and to the kernel functions chosen, it is especially sensitive to noise level estimates, and different kernels will lead to different predictions, so this technique is expected to work for only some of the measurements. The kernel and other parameters governing the behavior of the process were chosen by trial and error for one of the measurements and slightly modified to match others using visual assessment.

The output of the method provides an estimated mean and standard deviation of each point in the sample. This allows filtering outliers by comparing the difference of each point from the estimated mean and using the standard deviation as a local threshold. For this work, one standard deviation was used as a threshold to determine if a point is an outlier.

The visualization plot on the right shows three different graphical representations:

1. Points that pass the Gaussian process filter as blue points under the legend “Filtered GP”.
2. The Gaussian process predicated mean shown as black points that is easily visible in the plot since it is different than the original sample points. The legend label is “GP Prediction”
3. The Gaussian process predicated standard deviation shown as thin vertical bars labeled as “GP STD Span”

Note that this technique is different than other techniques used because it is based on prediction by regression rather than classification or filtering.

Filtering Using Support Vector Machine

A Support Vector Machine (SVM) is a classification technique that is based on finding the widest region separating classes of data. The simplest explanation in two dimensions to SVM is that it finds the farthest apart parallel lines that separate data classes. However, to cope with non linear type data, SVM allow introduction of a function that wraps space called Kernel. Popular kernels are Radial Based Functions (RBF). In this work a specific variation of SVM is used that has only a single class. Scikit-learn uses the OneClassSVM that is briefly described as novelty detection technique in http://scikit-learn.org/stable/modules/svm.html#svm-outlier-detection .

There is one parameter "nu" that governs the behavior of the SVN. It represents the upper bound for the proportion of training errors – in this case it was chosen to be 1% assuming that outliers are rare in the data. In this case, the training phase actually determines the outliers. Therefore if in the future very noisy data is introduced, this parameter may need adjustment.

The legend of the right panel shows SVN points that passed the filter as "Filtered SVM".

Filtering Using DBSCAN

This is a clustering algorithm that grows clusters from core high density regions and incrementally adds samples to it. This technique allows clusters that are non convex in shape and the idea to use it comes from the realization that outliers are many times far from the other data points, therefore a density sensitive algorithm should detect them.

This clustering algorithm is implemented in scikit-learn in the class DBSCAN that is described in this link: http://scikit-learn.org/stable/modules/clustering.html#dbscan . It is governed by several parameters where the main parameter "epsilon" defines the maximal distance between samples to be considered in the same neighborhood. This parameter is somewhat incompatible with the fact that we have several measurements with varying densities.

Therefore the DBSCAN algorithm was wrapped within a loop that changes the "epsilon" parameter by doubling it each loop iteration before executing the clustering algorithm. The loop stops where the number of clusters found is 3 or less. The idea is that outliers look like vertical spikes, so there should be two classes of outliers, above the main data series and below the main data series. The clusters detected are then inspected for their size to figure out which of them represent outliers. Outliers were defined after some testing as clusters less than 2% of the sample size.

This wrapping and outlier classification step is necessary to avoid cases where a single outlier point separates the data horizontally into three classes 1) points before the outlier, 2) the outlier point, 3) points after the outlier.

DBSCAN filtered points are marked on the right plot set as "Filtered DBSCAN" in cyan color.

Filtering Using Agglomerative Clustering

This algorithm uses Hierarchical clustering where each cluster starts with a single sample and clusters are merged together according to linkage criteria specified. Scikit-learn uses the method AgglomerativeClustering documented in http://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering . There are several parameters governing the convergence, the most important one is the number of classes to detect which defines a stop criteria for the recursive merging process. Another input parameter is the type of distance metric which was chosen as Manhattan distance in this work, after some trial and error. Another important parameter used is the linkage criterion. It was chosen to merge sets of clusters that minimize the average distance between set of clusters.

However, similar to DBSCAN, the algorithm may return large clusters that do not represent outliers, so another proportion criterion was added to detect the outlier clusters by comparing its proportion of the entire measurement sample to the proportion threshold. Clusters of less than 0.1 proportion of the number of points were considered outliers.

Agglomerative clustering filter is marked on the right panel plots with the legend title "Filtered AGG" in purple.

Filtering Using Local Outlier Factor

This unsupervised learning technique is based on sample density where each sample is compared to its neighbors, trying to isolate low density points as outliers. The scikit-learn implementation appears in the class LocalOutlierFactor with an example in http://scikit-learn.org/stable/auto_examples/neighbors/plot_lof.html . The method is documented in (Breunig et. al. 2000) . The technique uses efficient spacial representation techniques to speed up the calculations.

The techniques allows defining the number of neighbors which is typically unknown. The scikit-learn documentation recommends using the number 20, yet the number 10 seemed to work well enough in this work. Another influential factor is the type of metric used for distance calculations, which was chosen to be Manhattan distance to emphasize the vertical/horizontal component in samples that will otherwise smooth in euclidean geometry. The most important parameter is the contamination which defines the number of outlier points expected. The number 0.01 seemed to give reasonable results for contamination since only a very low number of outliers per measurements is expected.

Points filtered by local outlier factor are shown on the right side plots in pink color with the legend "Filtered LOF".

Filtering Using K Nearest Neighbors

The K nearest neighbors is a known problem in computer science trying to locate the K nearest samples closest to a certain sample. It is documented in http://scikit-learn.org/stable/modules/neighbors.html . This problem can be relatively efficiently solved using space partition techniques such as kd-tree or ball tree. The availability of an efficient implementation in scikit-learn in the form of the kneighbors_graph method makes it possible to create a simple filtering solution based on neighborhood.

The method returns the distance of each point from its K nearest neighbors in a matrix, summing a row of the matrix results in the sum of distances of neighbors for each point. Since the number of neighbors is preset, this number is proportional to the average distance of points from the sample point. We can therefore easily calculate the mean of distances of the K nearest neighbors for each point quite efficiently by summing the rows of the neighborhood matrix to create a vector of distance sums. We can then see for each point if that sum of distances is a statistical outlier by calculating mean and standard deviation of the vector and seeing if the difference between distances and the mean distance is above 3 standard deviations – this should capture 99.7% assuming normal distribution.

In this implementation the Manhattan distance was chosen as a metric to avoid the smoothing of euclidean space. This technique is efficient and filtered points using it are shown in yellow color in the plots to the right under the legend "Filtered KNN".

Filtering Using a Window

This simple technique uses the fact that the data is ordered along the time axis. It uses a window that looks back on neighboring 10 points and if the point value compared to the mean of the values of the previous points in the window deviates more than 6 standard deviations, then the point is considered an outlier. This simple idea was implemented to see how a very naive approach compares to some of the sophisticated approaches provided by machine learning techniques in scikit learn.

The method has an advantage that it is local and can be used with very few data points to predict the future behavior. The method seems to have some nice successes visually. However, in some cases it is quite different in its detections from other methods. It seems to detect the start of spikes in data as outliers, yet if a spike persists the mix/max points are no longer considered outliers since the method detects the persistence of the phenomenon.

The behavior of this method on the right panel is seen in light green color using the legend title "Filtered WIN".

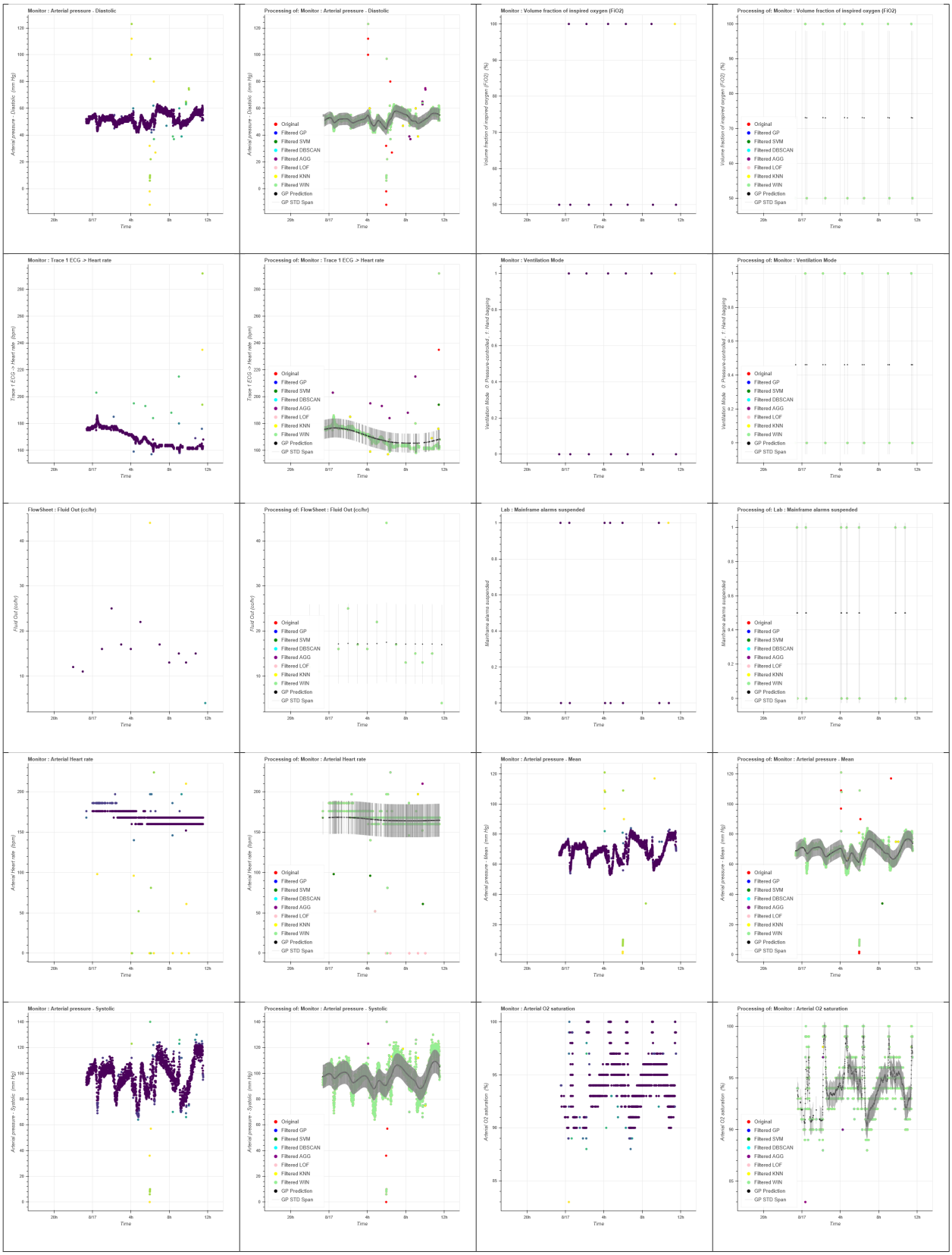

Figure 1. ICU data visualized for major measurements with filtering plot to the right.

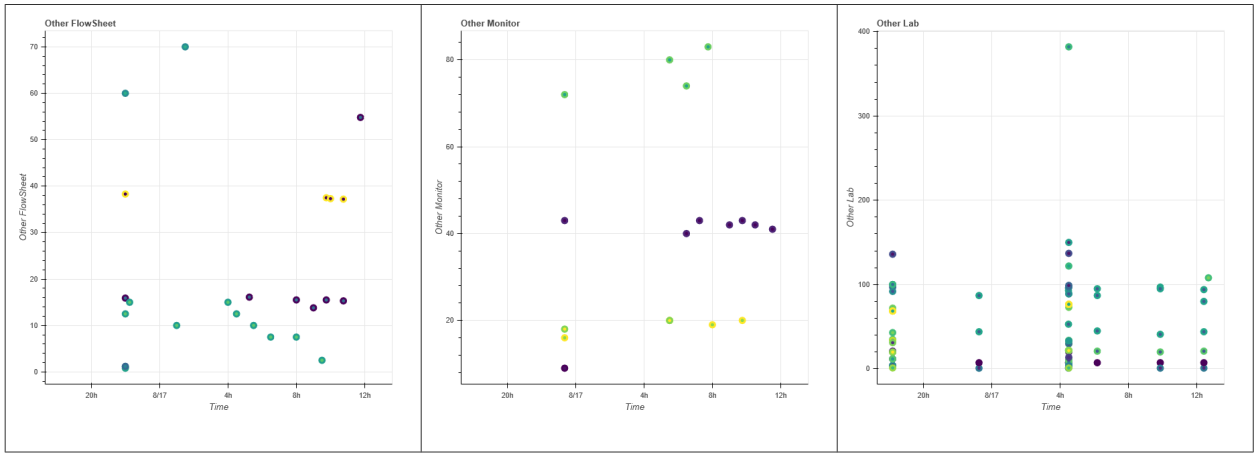

Figure 2. ICU data visualized for all remaining measurements: Flowsheet on the left, Monitor in the center, and Lab on the right . In the interactive version the user can hover over the points to get information on each measurement.

RESULTS

Figures 1-2 show the outputs produced by the visualization for the data set. Paper format does not allow good representation nor scrolling or interactive visualization, so the reader is encouraged to view the output on the web using the following link: http://sites.google.com/site/jacobbarhak/home/VisualICU.html

Figure 1 shows the major time series plots with their filtering plots to the right, Figure 2 shows the remaining data plots aggregated.

DISCUSSION

The visualization emphasizes the amount of information clinicians are exposed to. Even with regards to a single patient, the amount of information cannot be easily displayed in one location and interactive measures need to be added to allow data exploration. This amount of data also suggests that clinicians need help to quickly grasp the data, especially considering the pressure on the clinician that many times has to handle multiple patients and may not have sufficient time to study data in depth. Both machine learning algorithms and visual cues may be of great help to aid the clinician and therefore the patient.

In this work both these methods were used in very simple tasks of automated outlier filtering, yet the ideas presented can be used further. The concept followed was that no single algorithm is good enough to handle the data; an amalgamation of techniques is needed to provide a good picture of the data. In this work, a voting system allowed different methods to contribute to the picture of what is an outlier. The current output may visually assist a clinician yet cannot replace the decision making. Note that there was no clinical expert feedback to this work.

However, once such feedback is provided, it can open the door for more algorithms that will learn expert preference and be able to adjust the voting to emphasize methods that have better success in detecting the outliers. Moreover, recall that some outliers can be associated with some temporary equipment misuse such as a sensor being detached or some other interference that is not significant, while other outliers are significant and may indicate an important issue that needs clinical attention. Clinical feedback can also provide guidance to help figure out what points indicate this and an amalgamation of methods may have an advantage in such alerting systems.

Yet with advance in machine intelligence it will soon be possible to make advances towards possible prediction/projection of deterioration of a patient looking at data forms. Some of the techniques used here look back on data retrospectively while other techniques can more easily look forward at the next time step. Yet even

retrospective techniques can be used by using a window of data looking back in time to improve their responsiveness time. The size of the window may be adjusted or the system can try different window sizes and use the voting system and additional machine learning training to provide a good answer. This idea can be extended to parameters used by techniques in this work, the same method with different parameters can be used in creating more votes, thus increasing computational demands. Due to the cheap price and availability of computing power, trading of computational power vs. capabilities seems a fair trade and the balance can be decided per application.

Another possible extension to the techniques used here is correlating between different measurements for predictive purposes. Using multiple measurements may be useful in filtering data, yet it may have much more value in prediction/projection applications.

Regardless of the purpose of machine learning algorithm use, filtering, prediction/projection, or other analysis, the human clinician should still be in the loop for the foreseeable future. The purpose of machine analysis should be to support the clinician to increase productivity and quality, therefore helping the patients. However, such algorithms may quickly become a necessity due to the amounts of clinical data that are being accumulated by different clinical data sources.

REPRODUCIBILITY INFORMATION

Jupyter Notebook server version 5.0.0 on Anaconda 64 bit version 5.0.1 with Python 2.7.14 |Anaconda, Inc.| (default, Oct 15 2017, 03:34:40) [MSC v.1500 64 bit (AMD64)] , Pandas version 0.20.3, bokeh Version 0.12.10, scikit learn version 0.19.1. numpy version 1.13.3. The code was executed on Windows 10 machine. The execution code is available on Github at https://github.com/Jacob-Barhak/VisICU

ACKNOWLEDGEMENTS

Thanks to Tadashi Kamio for the email correspondence that resulted in this work.

REFERENCES

Bokeh (Online). https://Bokeh.pydata.org/en/latest/

Breunig, M. M., Kriegel, H. P., Ng, R. T., & Sander, J. (2000). LOF: identifying density-based local outliers. ACM sigmod record. Proc. ACM SIGMOD 2000 Int. Conf. On Management of Data, Dallas, TX, 2000

Celi, L.A. Mark, R. G. (2013). “Big Data” in the Intensive Care Unit. Closing the Data Loop. Am J Respir Crit Care Med. 2013 Jun 1; 187(11): 1157–1160. PMCID: PMC3734609. https://dx.doi.org/10.1164%2Frccm.201212-2311ED

Fonnesbeck, C. (2017). Fitting Gaussian Process Models in Python, Domino Blog. Online: https://blog.dominodatalab.com/fitting-gaussian-process-models-Python/

Gore, R. Chandra, M.S. (2015). "PACT: participant-centered clinical trial framework" Proceedings of the Conference on Summer Computer Simulation. https://dl.acm.org/citation.cfm?id=2874974

Ide, N.C. Loane, R.F. & Demner-Fushman, D. (2007). “Essie: A concept-based search engine for structured biomedical text”. J Am Med Inform Assoc. 14(3) pp. 253-63 https://doi.org/10.1197/jamia.M2233

Lichman, M. (2013). “UCI Machine Learning Repository” [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Olchanski, N. Lin, P. Winn, A. Lang, K. (2013). "Using EMR data for conducting retrospective studies: Opportunities and Pitfalls". ISPOR 2013, New Orleans, LA. Online: Accessed 14-Jan-2018 http://www.ispor.org/meetings/neworleans0513/releasedpresentations/W14_ALLSPEAKERS.pdf

Pedregosa, F, ,Varoquaux, G. Gramfort, A. Michel, V. Thirion, B. Grisel, O. Blondel, M. Prettenhofer, P. Weiss, R. Dubourg, V. Vanderplas, J. Passos, A. Cournapeau, D. Brucher, M. Perrot, M. Duchesnay E. (2011). Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830.

PhysioNet, MIMIC II (Online). https://physionet.org/mimic2/mimic2_access.shtml

PhysioNet, The MIMIC-III Clinical Database (Online). https://physionet.org/physiobank/database/mimic3cdb/

PhysioNet, The MIMIC Database (Online). https://physionet.org/physiobank/database/mimicdb/#complete-records

Royal College of Physicians (Online). "National Early Warning Score (NEWS) 2". https://www.rcplondon.ac.uk/projects/outputs/national-early-warning-score-news-2

scikit-learn (Online). Machine Learning in Python http://scikit-learn.org/stable/index.html

StarCluster: (Online). http://star.mit.edu/cluster/

Tufte, E. R. (1997). Visual Explanations: Images and Quantities, Evidence and Narrative. Graphics Press ISBN-13: 978-0961392123

UCI, Machine Learning Repository. ICU Dataset (Online) https://archive.ics.uci.edu/ml/datasets/ICU

Zarin, D.A., Tse, T. Williams, R. J. & Carr S. (2016). Trial Reporting in ClinicalTrials.gov — The Final Rule. N Engl J Med ; 375 pp. 1998-2004 http://dx.doi.org/10.1056/NEJMsr1611785